Programmation Impérative

PULUDISU Mpia Mimpiya
Numéro étudiant : 18913467
L1 Informatique

23 septembre 2024

Table des matières

1 Attendus

Pour ce cours, nous devons rendre les exercices de la série cx17 et les exercices cx25.0 et cx25.1 de la série cx25. Comme certains exercices dépendent d'autres exercices dans le cours, j'ai ajouté exercice cx15.3 de la série cx15 et les exercices cx18.1 et cx18.6 de la série cx18.

2 Série cx18

2.1 cx18.1 : Recoder la fonction putlist() pour afficher la liste en ordre inverse

```
int main ( int k ) // k utlis é e comme locale
{ list L = nil ;
    for ( k = 'a ' ; k < 'i ' ; k ++)
        L = cons (k , L ) ;
        putlist ( L ) ;
    return 0 ; }

list cons ( int car , list L )
{ list new = malloc ( sizeof ( node ) ) ;
if (! new ) usage ( " cons : manque de RAM " ) ; // enfin , un peu de s é
   rieux !
    new -> car = car ;
    new -> cdr = L ;
    return new ; }

void putlist ( list L )
    { if (! L ) return ; // nil : fin de liste
    printf ( " % c " , L -> car ) ;
    putlist ( L -> cdr ) ; }
```

Dans le programme ci-dessus, recoder la fonction putlist() pour qu'elle affiche la liste en ordre inverse, i.e. l'ordre dans lequel les éléments ont été rencontrés et empilés.

Pour afficher la liste en ordre inverse, nous devons inverser la liste avant de l'afficher, voici un petit programme qui réalise cette tâche :

```
/* ********************************************************
 * Nom            : cx18.1.c
 * Rôle           : Recoder la fonction putlist() pour afficher la liste en
    ordre inverse
 * Auteur         : Mpia Mimpiya PULUDISU
 * Version        : 1.0
 * Date           : 2024-08-26
 * Licence        : L1 PROGC
 * ********************************************************
 * Compilation    : gcc -W cx18.1.c -o main
 * Usage          : ./main
 * ********************************************************/

#include <stdio.h>
#include <stdlib.h>

// Définition d'une structure de noeud pour une liste chaînée
typedef struct node {
    int car;            // Valeur du noeud
    struct node *cdr;   // Pointeur vers le noeud suivant
} *list;                // Définition d'un type de pointeur vers un noeud,
   appelé 'list'

// Fonction pour créer un nouveau noeud et l'ajouter à la fin de la liste
list cons(int car, list L) {
    list new = malloc(sizeof(struct node)); // Allocation de mémoire pour un
        nouveau noeud
    if (!new) {                             // Vérification de l'allocation
       réussie
```

```
        fprintf(stderr, "cons : manque de RAM\n"); // Message d'erreur en
            cas d'échec
        exit(1);                            // Terminaison du programme en
            cas d'échec
    }
    new->car = car;    // Initialisation de la valeur du noeud
    new->cdr = NULL;   // Le pointeur 'cdr' est initialisé à NULL

    if (!L) {          // Si la liste est vide, retourne le nouveau noeud
        return new;
    }

    list temp = L;     // Temporaire pour parcourir la liste
    while (temp->cdr) { // Parcours jusqu'à la fin de la liste
        temp = temp->cdr;
    }
    temp->cdr = new;   // Ajout du nouveau noeud à la fin de la liste
    return L;          // Retourne la liste mise à jour
}

// Fonction pour afficher les éléments de la liste
void putlist(list L) {
    if (!L) return;              // Si la liste est vide, on retourne
    printf("%c ", L->car);       // Affichage de la valeur du noeud courant
    putlist(L->cdr);             // Appel récursif pour afficher le reste de
        la liste
}

// Fonction principale
int main() {
    list L = NULL;               // Initialisation de la liste à NULL (vide)
    int k;
    for (k = 'h'; k >= 'a'; k--) { // Boucle pour ajouter les caractères de
        'h' à 'a' dans la liste
        L = cons(k, L);          // Ajout de chaque caractère à la fin de la
            liste
    }
    putlist(L);                  // Affichage de la liste
    printf("\n");                // Ajout d'une nouvelle ligne pour une
        meilleure lisibilité
    return 0;                    // Fin du programme
}
```

Nous pouvons compiler et lancer le programme :

```
gcc -W cx18.1.c -o cx18.1
./cx18.1
```

2.2 cx18.6 : Déporter les fonctions de gestion de listes dans une bibliothèque dynamique

Adapter votre solution de [cx18.1] en déportant les fonctions de gestion de listes dans une bibliothèque dynamique vérifier que le programme tourne toujours.
Essayer d'adapter de la même façon les autres exemples de la série...

Pour déporter les fonctions de gestion de listes dans une bibliothèque dynamique, nous allons créer les fichiers suivants dans la même arborecence :

1. `list.h` : pour les déclarations. Voici le code de ce fichier :

```c
// Protection contre les inclusions multiples
//
/*
    La directive #ifndef en C et C++ signifie if not defined (si non dé
       fini).
    Elle est utilisée pour vérifier si un symbole (ou macro) na pas
       encore été défini.
    Si ce symbole nest pas défini, le code entre #ifndef et #endif sera
    inclus par le préprocesseur.
*/
#ifndef LIST_H
#define LIST_H

// Inclusion de la bibliothèque standard pour les fonctions utilitaires
    générales (malloc, free, etc.)
#include <stdlib.h>
// Définition de la structure de données pour une liste chaînée
typedef struct node {
    int car;  // Valeur de l'élément de la liste
    struct node *cdr;  // Pointeur vers le prochain élément de la liste
} *list;  // Définition du type 'list' comme un pointeur vers un 'node'

// Déclaration des fonctions pour manipuler la liste chaînée
list cons(int car, list L);  // Fonction pour ajouter un élément en tê
   te de liste
void putlist(list L);  // Fonction pour afficher les éléments de la
   liste
list reverse_list(list L);  // Fonction pour inverser l'ordre des élé
   ments de la liste

#endif  // Fin de la protection contre les inclusions multiples
```

2. `list.c` : pour les définitions. Voici le code de ce fichier :

```c
// Inclusion de la bibliothèque standard pour les entrées/sorties (
   printf, fprintf, etc.)
#include <stdio.h>
#include <stdlib.h>
// Inclusion du fichier d'en-tête "list.h" qui contient les définitions
    et déclarations pour les listes chaînées
#include "list.h"

// Fonction pour ajouter un élément en tête de liste
list cons(int car, list L) {
    list new = malloc(sizeof(struct node));  // Allouer de la mémoire
       pour un nouveau nud
    if (!new) {
        fprintf(stderr, "cons : manque de RAM\n");  // Afficher un
           message d'erreur si l'allocation échoue
        exit(1);  // Quitter le programme en cas d'erreur
    }
    new->car = car;  // Initialiser la valeur du nouveau nud
    new->cdr = L;  // Lier le nouveau nud à la liste existante
    return new;  // Retourner le nouveau nud
}

// Fonction pour afficher les éléments de la liste
void putlist(list L) {
    L = reverse_list(L);  // Inverser la liste pour l'affichage
    while (L != NULL) {
        printf("%c ", L->car);  // Afficher la valeur de chaque nud
        L = L->cdr;  // Passer au nud suivant
    }
    printf("\n");  // Nouvelle ligne après l'affichage de la liste
}

// Fonction pour inverser l'ordre des éléments de la liste
list reverse_list(list L) {
    list prev = NULL;  // Initialiser le pointeur vers le nud précédent
    list current = L;  // Initialiser le pointeur vers le nud actuel
    list next = NULL;  // Initialiser le pointeur vers le nud suivant

    while (current != NULL) {
        next = current->cdr;  // Sauvegarder le nud suivant
        current->cdr = prev;  // Inverser le lien du nud actuel
        prev = current;  // Avancer le pointeur vers le nud précédent
        current = next;  // Avancer le pointeur vers le nud actuel
    }
    return prev;  // Retourner le nouveau premier nud de la liste
       inversée
}
```

3. Et `main.c` : pour le programme principal. Voici le code de ce fichier :

```c
/* ******************************************************
 * Nom            : main.c
 * Rôle           : Déportation de fonctions de gestion de listes
 *                  dans une bibliothèque dynamique
 * Auteur         : Mpia Mimpiya PULUDISU
 * Version        : 1.0
 * Date           : 2024-08-26
 * Licence        : L1 PROGC
```

```
* ******************************************************
* Compilation   : gcc -o main main.c list.c
* Usage         : ./main
* ******************************************************/

// Inclusion du fichier d'en-tête "list.h" qui contient les dé
   finitions et déclarations pour les listes chaînées
#include "list.h"

int main() {
    list L = NULL;  // Initialiser la liste à NULL
    int k;
    // inversion des elements de la listes, on aura:  h g f e d c b a
    for (k = 'h'; k >= 'a'; k--) {
        L = cons(k, L);  // Ajouter chaque caractère de 'h' à 'a' en t
           ête de liste
    }
    putlist(L);  // Afficher les éléments de la liste
    return 0;  // Retourner 0 pour indiquer que le programme s'est
       terminé avec succès
}
```

Pour compiler et exécuter le programme,voici les étapes à suivres :

1. Compilation et création de la bibliothèque dynamique :
   - Compilez le fichier `list.c` pour créer un objet partagé :

     ```
     gcc -c -fPIC list.c -o list.o
     ```

   - Créez la bibliothèque dynamique à partir de l'objet partagé :

     ```
     gcc -shared -o liblist.so list.o
     ```

   - Compilez le programme principal en liant la bibliothèque dynamique :

     ```
     gcc -o main main.c -L. -llist
     ```

2. Exécuter le programme : Pour exécuter le programme, nous devons nous assurer que le système peut trouver la bibliothèque dynamique. Nous définissons la variable d'environnement `LD_LIBRARY_PATH` pour inclure le répertoire courant :

   ```
   export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
   ./main
   ```

Si toutes les étapes sont respectées, on devait avoir `h g f e d c b a`.

3 Série cx15

3.1 cx15.3 : Créer le fichier texte `cx15.3.data` et lire les données dans une table de chaînes

Voici le contenu du fichier `cx15.3.data` :

```
Joe Jack William Averell
```

notre petit programme :

```c
/* ********************************************************
 * Nom            : main.c
 * Rôle           : Lis les informations dans un fichier
 * Auteur         : Mpia Mimpiya PULUDISU
 * Version        : 1.0
 * Date           : 2024-08-26
 * Licence        : L1 PROGC
 * ********************************************************
 * Compilation    : gcc -Wall main.c -o main
 * Usage          : ./main

 * ********************************************************/


 #include <stdio.h>   // Inclusion de la bibliothèque standard d'entrée/
    sortie
 #include <stdlib.h>  // Inclusion de la bibliothèque standard pour la
    gestion de la mémoire
 #include <string.h>  // Inclusion de la bibliothèque pour la manipulation
    des chaînes de caractères

 #define MAX_NAMES 100        // Définition de la taille maximale du tableau
     de noms
 #define MAX_NAME_LENGTH 50   // Définition de la longueur maximale d'un nom

 int main() {
     FILE *file = fopen("cx15.3.data", "r");  // Ouverture du fichier
        contenant les noms
     if (!file) {  // Vérification de l'ouverture réussie du fichier
         fprintf(stderr, "Erreur lors de l'ouverture du fichier\n");  //
            Message d'erreur en cas d'échec
         return 1;  // Retourne 1 pour indiquer une erreur
     }

     char names[MAX_NAMES][MAX_NAME_LENGTH];  // Déclaration du tableau pour
         stocker les noms
     int count = 0;  // Initialisation du compteur de noms

     // Lecture des noms depuis le fichier jusqu'à la fin du fichier (EOF)
     // EOF = End Of File (fin de fichier).
     while (fscanf(file, "%s", names[count]) != EOF) {
         count++;  // Incrémentation du compteur de noms
     }

     fclose(file);  // Fermeture du fichier

     // Boucle à travers tous les noms lus
     for (int i = 0; i < count; i++) {
```

```
        printf("%s\n", names[i]);  // Affichage de chaque nom
    }

    return 0;  // Retourne 0 pour indiquer que le programme s'est terminé
        avec succès
}
```

La compilation et l'exécution du programme :

```
gcc -W main.c -o main
./main
```

La sortie du programme est :

```
Joe
Jack
William
Averell
```

4 Série cx17

4.1 cx17.0 : coder et compiler ce programme !

coder et compiler ce programme !; voyez-vous l'opportunité de rajouter d'autres caractères à la valeur de split_chars ? La tabulation !? Les symboles $< et >$!? Essayez-le sur un fichier html, et voyez ce que vous pouvez filtrer ainsi...

Voici le programme réalisant la tâche demandée :

```c
#include <stdio.h>
#include <stdlib.h>

// Fonction pour filtrer certains caractères d'un fichier d'entrée et écrire
    le résultat dans un fichier de sortie
void filter_chars(const char *input_file, const char *output_file) {
    // Ouvre le fichier d'entrée en mode lecture
    FILE *infile = fopen(input_file, "r");
    // Ouvre le fichier de sortie en mode écriture
    FILE *outfile = fopen(output_file, "w");

    // Vérifie si les fichiers ont été ouverts correctement
    if (infile == NULL || outfile == NULL) {
        perror("Error opening file"); // Affiche un message d'erreur si un
            fichier ne peut pas être ouvert
        exit(EXIT_FAILURE); // Quitte le programme avec un code d'erreur
    }

    char ch;
    // Lit chaque caractère du fichier d'entrée jusqu'à la fin du fichier (
        EOF)
    while ((ch = fgetc(infile)) != EOF) {
        // Si le caractère n'est pas une tabulation, ni '<', ni '>', il est
            écrit dans le fichier de sortie
        if (ch != '\t' && ch != '<' && ch != '>') {
            fputc(ch, outfile);
        }
    }

    // Ferme les fichiers d'entrée et de sortie
    fclose(infile);
    fclose(outfile);
}

int main() {
    // Appelle la fonction filter_chars avec "index.html" comme fichier d'
        entrée et "output.txt" comme fichier de sortie
    filter_chars("index.html", "output.txt");
    // Affiche un message indiquant que le filtrage est terminé
    printf("Filtrage terminé. Vérifier le fichier output.txt\n");
    return 0;
}
```

4.2 cx17.1 : Exploiter une STOPLIST

modifier la fonction ajoute_ref() pour qu'elle vérifie si la référence n'existe pas déjà, et tester le programme sur de gros fichiers (# 50 paragraphes) pour vérifier son fonctionnement.

Voici le programme réalisant la tâche demandée :

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_REFS 1000 // Définir une taille maximale pour les références
#define MAX_LINE_LENGTH 1000 // Définir une longueur maximale pour chaque
    ligne lue

// Structure pour stocker les références
typedef struct {
    char refs[MAX_REFS][100]; // Tableau de chaînes pour stocker les réfé
        rences
    int count; // Compteur de références
} RefList;

// Fonction pour vérifier si une référence existe déjà
int ref_exists(RefList *list, const char *ref) {
    for (int i = 0; i < list->count; i++) {
        if (strcmp(list->refs[i], ref) == 0) {
            return 1; // La référence existe déjà
        }
    }
    return 0; // La référence n'existe pas
}

// Fonction pour ajouter une référence si elle n'existe pas déjà
void ajoute_ref(RefList *list, const char *ref) {
    if (!ref_exists(list, ref)) {
        strcpy(list->refs[list->count], ref); // Copier la référence dans le
            tableau
        list->count++; // Incrémenter le compteur de références
    }
}

// Fonction pour traiter un fichier et ajouter ses lignes comme références
void process_file(const char *filename, RefList *list) {
    FILE *file = fopen(filename, "r"); // Ouvrir le fichier en mode lecture
    if (file == NULL) {
        perror("Error opening file"); // Afficher un message d'erreur si le
            fichier ne peut pas être ouvert
        exit(EXIT_FAILURE); // Quitter le programme avec un code d'erreur
    }

    char line[MAX_LINE_LENGTH];
    // Lire chaque ligne du fichier jusqu'à la fin du fichier (EOF)
    while (fgets(line, sizeof(line), file)) {
        line[strcspn(line, "\n")] = '\0'; // Enlever le caractère de
            nouvelle ligne
        ajoute_ref(list, line); // Ajouter la ligne comme référence
    }

    fclose(file); // Fermer le fichier
}
```

```c
int main() {
    RefList list = { .count = 0 }; // Initialiser la liste des références

    // Traiter le fichier avec 50 paragraphes
    process_file("fichier.txt", &list);

    // Afficher les références pour vérifier
    for (int i = 0; i < list.count; i++) {
        printf("Référence %d: %s\n", i + 1, list.refs[i]);
    }

    return 0;
}
```

4.3 cx17.2 : Utiliser la solution de [cx15.3] pour lire et exploiter une STOPLIST

Utiliser votre solution de [cx15.3] pour rajouter au programme la capacité de lire et d'exploiter une STOPLIST comme dans le prototype PYTHON.

Pour cette étape, nous allons lire une STOPLIST à partir d'un fichier et l'utiliser pour filtrer les mots lus dans cx15.3.data.

Voici les fichiers de notre arborecence :

```
~/Bureau/IMPERATIVE/codes/cx17/cx17.2 (0.083s)
tree

.
├── cx15.3.data
├── main.c
└── stoplist.txt

0 directories, 3 files
```

Voici le programme principal :

```c
/* ********************************************************
 * Nom            : main.c
 * Rôle           : Lis les mots dans un fichier et exclue les
 *                  mots figurant la stoplist
 * Auteur         : Mpia Mimpiya PULUDISU
 * Version        : 1.0
 * Date           : 2024-08-26
 * Licence        : L1 PROGC
 * ********************************************************
 * Compilation    : gcc -Wall main.c -o main
 * Usage          : ./main
 * ********************************************************/
#include <stdio.h>   // Inclusion de la bibliothèque standard d'entrée/
    sortie
#include <stdlib.h>  // Inclusion de la bibliothèque standard pour la
    gestion de la mémoire
#include <string.h>  // Inclusion de la bibliothèque pour la manipulation
    des chaînes de caractères

#define MAX_NAMES 100        // Définition de la taille maximale du tableau
    de noms
#define MAX_NAME_LENGTH 50   // Définition de la longueur maximale d'un nom
#define MAX_STOPLIST 100     // Définition de la taille maximale du tableau
    de mots à exclure
```

```c
// Fonction pour vérifier si un mot est dans la liste des mots à exclure
int is_in_stoplist(char *word, char stoplist[][MAX_NAME_LENGTH], int
    stoplist_count) {
    for (int i = 0; i < stoplist_count; i++) {  // Boucle à travers la liste
        des mots à exclure
        if (strcmp(word, stoplist[i]) == 0) {   // Comparaison du mot avec
            chaque mot de la liste
            return 1;  // Retourne 1 si le mot est trouvé dans la liste
        }
    }
    return 0;  // Retourne 0 si le mot n'est pas trouvé dans la liste
}

int main() {
    FILE *file = fopen("cx15.3.data", "r");  // Ouverture du fichier
        contenant les noms
    if (!file) {  // Vérification de l'ouverture réussie du fichier
        fprintf(stderr, "Erreur lors de l'ouverture du fichier\n");  //
            Message d'erreur en cas d'échec
        return 1;  // Retourne 1 pour indiquer une erreur
    }

    char names[MAX_NAMES][MAX_NAME_LENGTH];  // Déclaration du tableau pour
        stocker les noms
    int count = 0;  // Initialisation du compteur de noms

    // Lecture des noms depuis le fichier jusqu'à la fin du fichier (EOF)
    while (fscanf(file, "%s", names[count]) != EOF) {
        count++;  // Incrémentation du compteur de noms
    }

    fclose(file);  // Fermeture du fichier

    FILE *stoplist_file = fopen("stoplist.txt", "r");  // Ouverture du
        fichier contenant la liste des mots à exclure
    if (!stoplist_file) {  // Vérification de l'ouverture réussie du fichier
        fprintf(stderr, "Erreur lors de l'ouverture du fichier stoplist\n");
            // Message d'erreur en cas d'échec
        return 1;  // Retourne 1 pour indiquer une erreur
    }

    char stoplist[MAX_STOPLIST][MAX_NAME_LENGTH];  // Déclaration du tableau
        pour stocker les mots à exclure
    int stoplist_count = 0;  // Initialisation du compteur de mots à exclure

    // Lecture des mots à exclure depuis le fichier jusqu'à la fin du
        fichier (EOF)
    while (fscanf(stoplist_file, "%s", stoplist[stoplist_count]) != EOF) {
        stoplist_count++;  // Incrémentation du compteur de mots à exclure
    }

    fclose(stoplist_file);  // Fermeture du fichier

    // Boucle à travers tous les noms lus
    for (int i = 0; i < count; i++) {
        // Affichage du nom s'il n'est pas dans la liste des mots à exclure
        if (!is_in_stoplist(names[i], stoplist, stoplist_count)) {
            printf("%s\n", names[i]);  // Affichage du nom
        }
    }
```

```
    return 0;  // Retourne 0 pour indiquer que le programme s'est terminé
        avec succès
}
```

La compilation et l'exécution du programme :

```
gcc -W main.c -o main
./main
```

La sortie du programme est :

```
William
Averell
```

4.4 cx17.3 : adapter le programme [cx17.2] pour qu'il manipule une table de structures

adapter le programme [cx17.2] pour qu'il manipule une table de structures - en utilisant, bien sûr, le script [px17] ou [sx17] pour comparer les résultats.

Voici le programme réalisant la tâche demandée :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAMES 100
#define MAX_NAME_LENGTH 50
#define MAX_STOPLIST 100

// Définition de la structure pour les noms
struct Nom {
    char nom[MAX_NAME_LENGTH];
    char categorie[MAX_NAME_LENGTH];  // Champ supplémentaire pour la caté
        gorie
};

// Fonction pour vérifier si un mot est dans la liste des mots à exclure
int is_in_stoplist(char *word, char stoplist[][MAX_NAME_LENGTH], int
    stoplist_count) {
    for (int i = 0; i < stoplist_count; i++) {
        if (strcmp(word, stoplist[i]) == 0) {
            return 1;
        }
    }
    return 0;
}

int main() {
    FILE *file = fopen("cx15.3.data", "r");
    if (!file) {
        fprintf(stderr, "Erreur lors de l'ouverture du fichier\n");
        return 1;
    }

    struct Nom noms[MAX_NAMES];  // Tableau de structures pour stocker les
        noms et catégories
    int count = 0;
```

```
    // Lecture des noms et des catégories depuis le fichier
    while (fscanf(file, "%s %s", noms[count].nom, noms[count].categorie) !=
        EOF) {
        count++;
    }

    fclose(file);

    FILE *stoplist_file = fopen("stoplist.txt", "r");
    if (!stoplist_file) {
        fprintf(stderr, "Erreur lors de l'ouverture du fichier stoplist\n");
        return 1;
    }

    char stoplist[MAX_STOPLIST][MAX_NAME_LENGTH];
    int stoplist_count = 0;

    // Lecture des mots à exclure depuis le fichier
    while (fscanf(stoplist_file, "%s", stoplist[stoplist_count]) != EOF) {
        stoplist_count++;
    }

    fclose(stoplist_file);

    // Boucle à travers tous les noms lus
    for (int i = 0; i < count; i++) {
        // Affichage du nom et de sa catégorie s'il n'est pas dans la liste
            des mots à exclure
        if (!is_in_stoplist(noms[i].nom, stoplist, stoplist_count)) {
            printf("%s (%s)\n", noms[i].nom, noms[i].categorie);  //
                Affichage du nom avec la catégorie
        }
    }

    return 0;
}
```

4.5 cx17.4 : Trier la table mots avant d'appeler dump()

modifier le programme [cx17.3] pour trier la table mots avant d'appeler dump().
Voici le programme réalisant la tâche demandée :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAMES 100
#define MAX_NAME_LENGTH 50
#define MAX_STOPLIST 100

// Définition de la structure pour les noms
struct Nom {
    char nom[MAX_NAME_LENGTH];
    char categorie[MAX_NAME_LENGTH];  // Champ supplémentaire pour la caté
        gorie
};

// Fonction pour vérifier si un mot est dans la liste des mots à exclure
```

```c
int is_in_stoplist(char *word, char stoplist[][MAX_NAME_LENGTH], int
    stoplist_count) {
    for (int i = 0; i < stoplist_count; i++) {
        if (strcmp(word, stoplist[i]) == 0) {
            return 1;
        }
    }
    return 0;
}

// Fonction de comparaison pour trier les noms par ordre alphabétique
int comparerNoms(const void *a, const void *b) {
    struct Nom *nomA = (struct Nom *)a;
    struct Nom *nomB = (struct Nom *)b;
    return strcmp(nomA->nom, nomB->nom);
}

int main() {
    FILE *file = fopen("cx15.3.data", "r");
    if (!file) {
        fprintf(stderr, "Erreur lors de l'ouverture du fichier\n");
        return 1;
    }

    struct Nom noms[MAX_NAMES];  // Tableau de structures pour stocker les
        noms et catégories
    int count = 0;

    // Lecture des noms et des catégories depuis le fichier
    while (fscanf(file, "%s %s", noms[count].nom, noms[count].categorie) !=
        EOF) {
        count++;
    }

    fclose(file);

    FILE *stoplist_file = fopen("stoplist.txt", "r");
    if (!stoplist_file) {
        fprintf(stderr, "Erreur lors de l'ouverture du fichier stoplist\n");
        return 1;
    }

    char stoplist[MAX_STOPLIST][MAX_NAME_LENGTH];
    int stoplist_count = 0;

    // Lecture des mots à exclure depuis le fichier
    while (fscanf(stoplist_file, "%s", stoplist[stoplist_count]) != EOF) {
        stoplist_count++;
    }

    fclose(stoplist_file);

    // Trier la table des noms avant de faire quoi que ce soit
    qsort(noms, count, sizeof(struct Nom), comparerNoms);

    // Boucle à travers tous les noms lus
    for (int i = 0; i < count; i++) {
        // Affichage du nom et de sa catégorie s'il n'est pas dans la liste
            des mots à exclure
        if (!is_in_stoplist(noms[i].nom, stoplist, stoplist_count)) {
            printf("%s (%s)\n", noms[i].nom, noms[i].categorie);  //
```

```
                Affichage du nom avec la catégorie
            }
        }

        return 0;
}
```

Cette version trie les noms alphabétiquement avant de les afficher en excluant ceux qui se trouvent dans la stoplist.

4.6 cx17.5 : modifier le programme réalisé pour [cx17.4] de sorte que les références soient gérées sous forme de listes

modifier le programme réalisé pour [cx17.4] de sorte que les références soient gérées sous forme de listes et que la référence courante ne soit pas rajoutée si elle est déjà présente dans la liste des références pour ce mot.

Voici le programme réalisant la tâche demandée :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAMES 100
#define MAX_NAME_LENGTH 50
#define MAX_REFERENCES 100  // Nombre maximum de références par mot
#define MAX_STOPLIST 100

// Définition de la structure pour les noms et leurs références
struct Nom {
    char nom[MAX_NAME_LENGTH];
    char categorie[MAX_NAME_LENGTH];
    char references[MAX_REFERENCES][MAX_NAME_LENGTH];  // Tableau de référ
       ences
    int ref_count;  // Compteur du nombre de références
};

// Fonction pour vérifier si une référence existe déjà dans la liste des réf
   érences d'un mot
int reference_exists(struct Nom *n, char *reference) {
    for (int i = 0; i < n->ref_count; i++) {
        if (strcmp(n->references[i], reference) == 0) {
            return 1;  // La référence est déjà présente
        }
    }
    return 0;  // La référence n'est pas trouvée
}

// Fonction pour ajouter une référence à un mot, si elle n'existe pas encore
void ajouter_reference(struct Nom *n, char *reference) {
    if (!reference_exists(n, reference)) {  // Si la référence n'existe pas
        strcpy(n->references[n->ref_count], reference);  // Ajout de la référ
           ence
        n->ref_count++;  // Incrémentation du compteur de références
    }
}

// Fonction pour vérifier si un mot est dans la liste des mots à exclure
int is_in_stoplist(char *word, char stoplist[][MAX_NAME_LENGTH], int
   stoplist_count) {
```

```
    for (int i = 0; i < stoplist_count; i++) {
        if (strcmp(word, stoplist[i]) == 0) {
            return 1;
        }
    }
    return 0;
}

int main() {
    FILE *file = fopen("cx15.3.data", "r");
    if (!file) {
        fprintf(stderr, "Erreur lors de l'ouverture du fichier\n");
        return 1;
    }

    struct Nom noms[MAX_NAMES];  // Tableau de structures pour stocker les
       noms et catégories
    int count = 0;

    // Lecture des noms et des catégories depuis le fichier
    while (fscanf(file, "%s %s", noms[count].nom, noms[count].categorie) !=
       EOF) {
        noms[count].ref_count = 0;  // Initialisation du compteur de référé
           rences à 0
        count++;
    }

    fclose(file);

    FILE *stoplist_file = fopen("stoplist.txt", "r");
    if (!stoplist_file) {
        fprintf(stderr, "Erreur lors de l'ouverture du fichier stoplist\n");
        return 1;
    }

    char stoplist[MAX_STOPLIST][MAX_NAME_LENGTH];
    int stoplist_count = 0;

    // Lecture des mots à exclure depuis le fichier
    while (fscanf(stoplist_file, "%s", stoplist[stoplist_count]) != EOF) {
        stoplist_count++;
    }

    fclose(stoplist_file);

    // Boucle à travers tous les noms lus
    for (int i = 0; i < count; i++) {
        // Si le mot n'est pas dans la liste des mots à exclure
        if (!is_in_stoplist(noms[i].nom, stoplist, stoplist_count)) {
            printf("Nom : %s, Catégorie : %s\n", noms[i].nom, noms[i].
               categorie);

            // Simulation de l'ajout de références pour chaque nom
            ajouter_reference(&noms[i], "ref1");
            ajouter_reference(&noms[i], "ref2");
            ajouter_reference(&noms[i], "ref1");  // Tentative de doublon

            // Affichage des références pour ce nom
            printf("Références : ");
            for (int j = 0; j < noms[i].ref_count; j++) {
                printf("%s ", noms[i].references[j]);
```

```
            }
            printf("\n");
        }
    }

    return 0;
}
```

Ce programme modifié gère les références sous forme de listes pour chaque mot et évite les doublons lors de l'ajout des références.

4.7 cx17.6 : Utiliser une STOPLIST représentée en interne sous la forme d'une liste élastique

Utiliser votre solution de [cx15.3] et de [cx17.2] pour ajouter au programme la capacité d'accueillir et d'exploiter une STOPLIST représentée en interne sous la forme d"une liste élastique, et lue à partir d'un fichier dont le nom sera spécifié sur la LdC s'il ne l'est pas, le programme ouvrira un fichier par défaut.

Pour cet exercice, nous allons utiliser une liste chaînée pour représenter la STOPLIST.

Pour réaliser cet exercice, voici les fichiers de notre arborecence :

```
~/Bureau/IMPERATIVE/codes/cx17/cx17.7 (0.08s)
tree

.
├── cx15.3.data
├── list.c
├── list.h
├── main.c
└── stoplist.txt

0 directories, 5 files
```

Nous allons nous attarder sur les trois fichiers principaux :

1. `list.h` :

```c
#ifndef LIST_H  // Si LIST_H n'est pas défini, alors définir LIST_H
#define LIST_H  // Définition de LIST_H pour éviter les inclusions
    multiples

// Définition de la structure de noeud pour une liste chaînée
typedef struct node {
    char *car;          // Pointeur vers une chaîne de caractères (
        valeur du noeud)
    struct node *cdr;   // Pointeur vers le noeud suivant
} *list;                // Définition d'un type de pointeur vers un
    noeud, appelé 'list'

// Déclaration de la fonction pour créer un nouveau noeud et l'ajouter
    en tête de liste
list cons(char *car, list L);

// Déclaration de la fonction pour afficher les éléments de la liste
void putlist(list L);

// Déclaration de la fonction pour vérifier si un mot est dans la liste
int is_in_list(list L, char *word);

#endif  // Fin de la conditionnelle #ifndef LIST_H
```

2. `list.c` :

```c
#include <stdio.h>   // Inclusion de la bibliothèque standard d'entrée/
    sortie
#include <stdlib.h>  // Inclusion de la bibliothèque standard pour la
    gestion de la mémoire
#include <string.h>  // Inclusion de la bibliothèque pour la
    manipulation des chaînes de caractères
#include "list.h"    // Inclusion du fichier d'en-tête "list.h"
    contenant les déclarations de la liste chaînée
```

```c
// Fonction pour créer un nouveau noeud et l'ajouter en tête de liste
list cons(char *car, list L) {
    list new = malloc(sizeof(struct node));  // Allocation de mémoire
       pour un nouveau noeud
    if (!new) {  // Vérification de l'allocation réussie
        fprintf(stderr, "cons : manque de RAM\n");  // Message d'erreur
            en cas d'échec
        exit(1);  // Terminaison du programme en cas d'échec
    }
    new->car = strdup(car);  // Copie de la chaîne de caractères dans
       le nouveau noeud
    new->cdr = L;  // Le pointeur 'cdr' pointe vers l'ancienne tête de
       liste
    return new;  // Retourne le nouveau noeud, qui est maintenant la tê
       te de liste
}

// Fonction pour afficher les éléments de la liste
void putlist(list L) {
    if (!L) return;  // nil : fin de liste
    putlist(L->cdr);  // Appel récursif pour afficher le reste de la
       liste
    printf("%s ", L->car);  // Affichage de la valeur du noeud courant
}

// Fonction pour vérifier si un mot est dans la liste
int is_in_list(list L, char *word) {
    while (L) {  // Boucle à travers la liste
        if (strcmp(L->car, word) == 0) {  // Comparaison du mot avec la
            valeur du noeud courant
            return 1;  // Retourne 1 si le mot est trouvé dans la liste
        }
        L = L->cdr;  // Passe au noeud suivant
    }
    return 0;  // Retourne 0 si le mot n'est pas trouvé dans la liste
}
```

3. main.c :

```c
/* ********************************************************
 * Nom            : main.c
 * Rôle           : Lis les mots dans un fichier et exclue les
 *                  mots figurant la stoplist
 * Auteur         : Mpia Mimpiya PULUDISU
 * Version        : 1.0
 * Date           : 2024-08-26
 * Licence        : L1 PROGC
 * ********************************************************
 * Compilation    : gcc -c list.c -o list.o
                    gcc -c main.c -o main.o
                    gcc main.o list.o -o main

                    OU
                    gcc -W main.c list.c -o main

 * Usage          : ./main
 * ********************************************************/


 #include <stdio.h>   // Inclusion de la bibliothèque standard d'entrée
```

```
    /sortie
#include <stdlib.h>  // Inclusion de la bibliothèque standard pour la
    gestion de la mémoire
#include <string.h>  // Inclusion de la bibliothèque pour la
    manipulation des chaînes de caractères
#include "list.h"    // Inclusion du fichier d'en-tête "list.h"
    contenant les déclarations de la liste chaînée

#define MAX_NAMES 100        // Définition de la taille maximale du
    tableau de noms
#define MAX_NAME_LENGTH 50   // Définition de la longueur maximale d'
    un nom

int main(int argc, char *argv[]) {
    FILE *file = fopen("cx15.3.data", "r");  // Ouverture du fichier
        contenant les noms
    if (!file) {  // Vérification de l'ouverture réussie du fichier
        fprintf(stderr, "Erreur lors de l'ouverture du fichier\n");
            // Message d'erreur en cas d'échec
        return 1;  // Retourne 1 pour indiquer une erreur
    }

    char names[MAX_NAMES][MAX_NAME_LENGTH];  // Déclaration du tableau
        pour stocker les noms
    int count = 0;  // Initialisation du compteur de noms

    // Lecture des noms depuis le fichier jusqu'à la fin du fichier (
        EOF)
    while (fscanf(file, "%s", names[count]) != EOF) {
        count++;  // Incrémentation du compteur de noms
    }

    fclose(file);  // Fermeture du fichier

    // Détermination du nom du fichier de la liste des mots à exclure
    const char *stoplist_filename = (argc > 1) ? argv[1] : "stoplist.
        txt";
    FILE *stoplist_file = fopen(stoplist_filename, "r");  // Ouverture
        du fichier contenant la liste des mots à exclure
    if (!stoplist_file) {  // Vérification de l'ouverture réussie du
        fichier
        fprintf(stderr, "Erreur lors de l'ouverture du fichier
            stoplist\n");  // Message d'erreur en cas d'échec
        return 1;  // Retourne 1 pour indiquer une erreur
    }

    list stoplist = NULL;  // Initialisation de la liste des mots à
        exclure à NULL (vide)
    char word[MAX_NAME_LENGTH];  // Déclaration d'un tableau pour
        stocker chaque mot lu

    // Lecture des mots à exclure depuis le fichier jusqu'à la fin du
        fichier (EOF)
    while (fscanf(stoplist_file, "%s", word) != EOF) {
        stoplist = cons(word, stoplist);  // Ajout de chaque mot à la
            liste des mots à exclure
    }

    fclose(stoplist_file);  // Fermeture du fichier

    // Boucle à travers tous les noms lus
```

```c
    for (int i = 0; i < count; i++) {
        // Affichage du nom s'il n'est pas dans la liste des mots à
           exclure
        if (!is_in_list(stoplist, names[i])) {
            printf("%s\n", names[i]);  // Affichage du nom
        }
    }

    return 0;  // Retourne 0 pour indiquer que le programme s'est
       terminé avec succès
}
```

La compilation et l'exécution du programme :

```
gcc -c list.c -o list.o
gcc -c main.c -o main.o
gcc main.o list.o -o main
./main
```

La sortie du programme, on a le résultat suivant en prenant compte de notre `cx15.3.data` et de notre `stoplist.txt` :

```
William
Averell
```

4.8 cx17.7 : Adapter la solution de [cx17.6] pour utiliser la bibliothèque créée pour [cx18.6]

Adapter votre solution de [cx17.6] pour utiliser la bibliothèque créée pour [cx18.6].

Pour cette étape, nous allons utiliser la bibliothèque créée pour [cx18.6].

Cet exercice est presque similaire à [cx17.6], nous allons donc aller rapidement :

Voici les codes de fichiers principaux :

1. `list.h` :

```c
#ifndef LIST_H  // Si LIST_H n'est pas défini, alors définir LIST_H
#define LIST_H  // Définition de LIST_H pour éviter les inclusions
    multiples

// Définition de la structure de noeud pour une liste chaînée
typedef struct node {
    char *car;          // Pointeur vers une chaîne de caractères (
        valeur du noeud)
    struct node *cdr;   // Pointeur vers le noeud suivant
} *list;                // Définition d'un type de pointeur vers un
    noeud, appelé 'list'

// Déclaration de la fonction pour créer un nouveau noeud et l'ajouter
    en tête de liste
list cons(char *car, list L);

// Déclaration de la fonction pour afficher les éléments de la liste
void putlist(list L);

// Déclaration de la fonction pour vérifier si un mot est dans la liste
int is_in_list(list L, char *word);

#endif  // Fin de la conditionnelle #ifndef LIST_H
```

2. `list.c` :

```c
#include <stdio.h>   // Inclusion de la bibliothèque standard d'entrée/
    sortie
#include <stdlib.h>  // Inclusion de la bibliothèque standard pour la
    gestion de la mémoire
#include <string.h>  // Inclusion de la bibliothèque pour la
    manipulation des chaînes de caractères
#include "list.h"    // Inclusion du fichier d'en-tête "list.h"
    contenant les déclarations de la liste chaînée

// Fonction pour créer un nouveau noeud et l'ajouter en tête de liste
list cons(char *car, list L) {
    list new = malloc(sizeof(struct node));  // Allocation de mémoire
        pour un nouveau noeud
    if (!new) {  // Vérification de l'allocation réussie
        fprintf(stderr, "cons : manque de RAM\n");  // Message d'erreur
            en cas d'échec
        exit(1);  // Terminaison du programme en cas d'échec
    }
    new->car = strdup(car);  // Copie de la chaîne de caractères dans
        le nouveau noeud
    new->cdr = L;  // Le pointeur 'cdr' pointe vers l'ancienne tête de
        liste
```

```
    return new;  // Retourne le nouveau noeud, qui est maintenant la tê
        te de liste
}

// Fonction pour afficher les éléments de la liste
void putlist(list L) {
    if (!L) return;  // nil : fin de liste
    putlist(L->cdr);  // Appel récursif pour afficher le reste de la
        liste
    printf("%s ", L->car);  // Affichage de la valeur du noeud courant
}

// Fonction pour vérifier si un mot est dans la liste
int is_in_list(list L, char *word) {
    while (L) {  // Boucle à travers la liste
        if (strcmp(L->car, word) == 0) {  // Comparaison du mot avec la
            valeur du noeud courant
            return 1;  // Retourne 1 si le mot est trouvé dans la liste
        }
        L = L->cdr;  // Passe au noeud suivant
    }
    return 0;  // Retourne 0 si le mot n'est pas trouvé dans la liste
}
```

3. main.c :

```
/* ********************************************************
 * Nom            : main.c
 * Rôle           : Lis les mots dans un fichier et exclue les
 *                  mots figurant la stoplist (adaptation de cx15.3)
 * Auteur         : Mpia Mimpiya PULUDISU
 * Version        : 1.0
 * Date           : 2024-08-26
 * Licence        : L1 PROGC
 * ********************************************************
 * Compilation    : gcc -c -fPIC list.c -o list.o
                    gcc -shared -o list.so list.o
                    gcc -o main main.c -L. -llist

                    OU

                    gcc -W main.c list.c
                    ./main

 * Usage          : export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
 *                  ./main
 * ********************************************************/

 #include <stdio.h>   // Inclusion de la bibliothèque standard d'entrée
    /sortie
 #include <stdlib.h>  // Inclusion de la bibliothèque standard pour la
    gestion de la mémoire
 #include <string.h>  // Inclusion de la bibliothèque pour la
    manipulation des chaînes de caractères
 #include "list.h"    // Inclusion du fichier d'en-tête "list.h"
    contenant les déclarations de la liste chaînée

 #define MAX_NAMES 100        // Définition de la taille maximale du
    tableau de noms
 #define MAX_NAME_LENGTH 50   // Définition de la longueur maximale d'
```

```
    un nom

int main(int argc, char *argv[]) {
    FILE *file = fopen("cx15.3.data", "r");  // Ouverture du fichier
        contenant les noms
    if (!file) {  // Vérification de l'ouverture réussie du fichier
        fprintf(stderr, "Erreur lors de l'ouverture du fichier\n");
            // Message d'erreur en cas d'échec
        return 1;  // Retourne 1 pour indiquer une erreur
    }

    char names[MAX_NAMES][MAX_NAME_LENGTH];  // Déclaration du tableau
        pour stocker les noms
    int count = 0;  // Initialisation du compteur de noms

    // Lecture des noms depuis le fichier jusqu'à la fin du fichier (
        EOF)
    while (fscanf(file, "%s", names[count]) != EOF) {
        count++;  // Incrémentation du compteur de noms
    }

    fclose(file);  // Fermeture du fichier

    // Détermination du nom du fichier de la liste des mots à exclure
    const char *stoplist_filename = (argc > 1) ? argv[1] : "stoplist.
        txt";
    FILE *stoplist_file = fopen(stoplist_filename, "r");  // Ouverture
        du fichier contenant la liste des mots à exclure
    if (!stoplist_file) {  // Vérification de l'ouverture réussie du
        fichier
        fprintf(stderr, "Erreur lors de l'ouverture du fichier
            stoplist\n");  // Message d'erreur en cas d'échec
        return 1;  // Retourne 1 pour indiquer une erreur
    }

    list stoplist = NULL;  // Initialisation de la liste des mots à
        exclure à NULL (vide)
    char word[MAX_NAME_LENGTH];  // Déclaration d'un tableau pour
        stocker chaque mot lu

    // Lecture des mots à exclure depuis le fichier jusqu'à la fin du
        fichier (EOF)
    while (fscanf(stoplist_file, "%s", word) != EOF) {
        stoplist = cons(word, stoplist);  // Ajout de chaque mot à la
            liste des mots à exclure
    }

    fclose(stoplist_file);  // Fermeture du fichier

    // Boucle à travers tous les noms lus
    for (int i = 0; i < count; i++) {
        // Affichage du nom s'il n'est pas dans la liste des mots à
            exclure
        if (!is_in_list(stoplist, names[i])) {
            printf("%s\n", names[i]);  // Affichage du nom
        }
    }

    return 0;  // Retourne 0 pour indiquer que le programme s'est
        terminé avec succès
}
```

La compilation et l'exécution du programme, voici les étapes à suivre :

1. **Compilation et création de la bibliothèque dynamique**
   (a) Compilez le fichier `list.c` pour créer un objet partagé :

```
gcc -c -fPIC list.c -o list.o
```

   (b) Créez la bibliothèque dynamique à partir de l'objet partagé :

```
gcc -shared -o list.so list.o
```

   (c) Compilez le programme principal en liant la bibliothèque dynamique :

```
gcc -o main main.c -L. -llist
```

2. **Exécution du programme**

```
export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
./main
```

4.9 cx17.8 : Adapter la solution de [cx17.6] pour utiliser la bibliothèque créée pour [cx18.7]

1. `list.h` :

```c
#ifndef LIST_H  // Si LIST_H n'est pas défini, alors définir LIST_H
#define LIST_H  // Définition de LIST_H pour éviter les inclusions
    multiples

// Définition de la structure de noeud pour une liste chaînée
typedef struct node {
    char *car;          // Pointeur vers une chaîne de caractères (
        valeur du noeud)
    struct node *cdr;   // Pointeur vers le noeud suivant
} *list;                // Définition d'un type de pointeur vers un
    noeud, appelé 'list'

// Déclaration de la fonction pour créer un nouveau noeud et l'ajouter
    en tête de liste
list cons(char *car, list L);

// Déclaration de la fonction pour afficher les éléments de la liste
void putlist(list L);

// Déclaration de la fonction pour vérifier si un mot est dans la liste
int is_in_list(list L, char *word);

#endif  // Fin de la conditionnelle #ifndef LIST_H
```

2. `list.c` :

```c
#include <stdio.h>   // Inclusion de la bibliothèque standard d'entrée/
    sortie
#include <stdlib.h>  // Inclusion de la bibliothèque standard pour la
    gestion de la mémoire
#include <string.h>  // Inclusion de la bibliothèque pour la
    manipulation des chaînes de caractères
```

```c
#include "list.h"    // Inclusion du fichier d'en-tête "list.h"
   contenant les déclarations de la liste chaînée

// Fonction pour créer un nouveau noeud et l'ajouter en tête de liste
list cons(char *car, list L) {
    list new = malloc(sizeof(struct node));  // Allocation de mémoire
       pour un nouveau noeud
    if (!new) {  // Vérification de l'allocation réussie
        fprintf(stderr, "cons : manque de RAM\n");  // Message d'erreur
            en cas d'échec
        exit(1);  // Terminaison du programme en cas d'échec
    }
    new->car = strdup(car);  // Copie de la chaîne de caractères dans
       le nouveau noeud
    new->cdr = L;  // Le pointeur 'cdr' pointe vers l'ancienne tête de
       liste
    return new;  // Retourne le nouveau noeud, qui est maintenant la tê
       te de liste
}

// Fonction pour afficher les éléments de la liste
void putlist(list L) {
    if (!L) return;  // nil : fin de liste
    putlist(L->cdr);  // Appel récursif pour afficher le reste de la
       liste
    printf("%s ", L->car);  // Affichage de la valeur du noeud courant
}

// Fonction pour vérifier si un mot est dans la liste
int is_in_list(list L, char *word) {
    while (L) {  // Boucle à travers la liste
        if (strcmp(L->car, word) == 0) {  // Comparaison du mot avec la
            valeur du noeud courant
            return 1;  // Retourne 1 si le mot est trouvé dans la liste
        }
        L = L->cdr;  // Passe au noeud suivant
    }
    return 0;  // Retourne 0 si le mot n'est pas trouvé dans la liste
}
```

3. main.c :

```c
/* ******************************************************
 * Nom            : main.c
 * Rôle           : Lis les mots dans un fichier et exclue les
 *                  mots figurant la stoplist (adaptation de cx17.6)
 * Auteur         : Mpia Mimpiya PULUDISU
 * Version        : 1.0
 * Date           : 2024-08-26
 * Licence        : L1 PROGC
 * ******************************************************
 * Compilation    : gcc -c -fPIC list.c -o list.o
                    gcc -shared -o list.so list.o
                    gcc -o main main.c -L. -llist

                    gcc -W main.c list.c main
                    ./main

 * Usage          : export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
 *                  ./main
```

```c
* ******************************************************/
#include <stdio.h>   // Inclusion de la bibliothèque standard d'entrée/sortie
#include <stdlib.h>  // Inclusion de la bibliothèque standard pour la gestion de la mémoire
#include <string.h>  // Inclusion de la bibliothèque pour la manipulation des chaînes de caractères
#include "list.h"    // Inclusion du fichier d'en-tête "list.h" contenant les déclarations de la liste chaînée

#define MAX_NAMES 100        // Définition de la taille maximale du tableau de noms
#define MAX_NAME_LENGTH 50   // Définition de la longueur maximale d'un nom

int main(int argc, char *argv[]) {
    FILE *file = fopen("cx15.3.data", "r");  // Ouverture du fichier contenant les noms
    if (!file) {  // Vérification de l'ouverture réussie du fichier
        fprintf(stderr, "Erreur lors de l'ouverture du fichier\n");
            // Message d'erreur en cas d'échec
        return 1;  // Retourne 1 pour indiquer une erreur
    }

    char names[MAX_NAMES][MAX_NAME_LENGTH];  // Déclaration du tableau pour stocker les noms
    int count = 0;  // Initialisation du compteur de noms

    // Lecture des noms depuis le fichier jusqu'à la fin du fichier (EOF)
    while (fscanf(file, "%s", names[count]) != EOF) {
        count++;  // Incrémentation du compteur de noms
    }

    fclose(file);  // Fermeture du fichier

    // Détermination du nom du fichier de la liste des mots à exclure
    const char *stoplist_filename = (argc > 1) ? argv[1] : "stoplist.txt";
    FILE *stoplist_file = fopen(stoplist_filename, "r");  // Ouverture du fichier contenant la liste des mots à exclure
    if (!stoplist_file) {  // Vérification de l'ouverture réussie du fichier
        fprintf(stderr, "Erreur lors de l'ouverture du fichier stoplist\n");  // Message d'erreur en cas d'échec
        return 1;  // Retourne 1 pour indiquer une erreur
    }

    list stoplist = NULL;  // Initialisation de la liste des mots à exclure à NULL (vide)
    char word[MAX_NAME_LENGTH];  // Déclaration d'un tableau pour stocker chaque mot lu

    // Lecture des mots à exclure depuis le fichier jusqu'à la fin du fichier (EOF)
    while (fscanf(stoplist_file, "%s", word) != EOF) {
        stoplist = cons(word, stoplist);  // Ajout de chaque mot à la liste des mots à exclure
    }

    fclose(stoplist_file);  // Fermeture du fichier
```

```
    // Boucle à travers tous les noms lus
    for (int i = 0; i < count; i++) {
        // Affichage du nom s'il n'est pas dans la liste des mots à
           exclure
        if (!is_in_list(stoplist, names[i])) {
            printf("%s\n", names[i]);  // Affichage du nom
        }
    }

    return 0;  // Retourne 0 pour indiquer que le programme s'est
       terminé avec succès
}
```

Pour compiler et exécuter le programme, voir l'exercice cx17.7 c'est quasi le même processus.

5 Série cx25

5.1 cx25.0 : Programmer cet émulateur de sorte qu'il puisse lire le code numérique (ou symbolique) à partir d'un fichier

Ce petit programme écrit en C simule un processeur simple avec une mémoire de taille fixe et un ensemble d'instructions de base. Il charge un programme à partir d'un fichier (program.hex), l'exécute et affiche l'état de l'accumulateur (A) et du compteur de programme (PC) à chaque étape.

```
/*
Ce petit programme simule un processeur avec une mémoire de
taille fixe. Il lit un fichier contenant des instructions et exécute ces
instructions, affichant les états de l'accumulateur (A) et du compteur de
programme (PC).

compilation: gcc -W cx25.0.c -o main
             ./main program.hex
*/

// Inclusion de la bibliothèque standard pour les entrées/sorties (printf,
   scanf, etc.)
#include <stdio.h>
// Inclusion de la bibliothèque standard pour les fonctions utilitaires géné
   rales (malloc, free, exit, etc.)
#include <stdlib.h>
 // Inclusion de la bibliothèque pour les opérations sur les chaînes de
    caractères (memset, memcpy, etc.)
#include <string.h>

// Définir la taille de la mémoire et l'adresse de début du programme
#define MEMORY_SIZE 256
#define PROGRAM_START 0x50  // Adresse de début du programme

// Déclaration de la mémoire et des registres
unsigned char memory[MEMORY_SIZE];  // Tableau représentant la mémoire
unsigned char A = 0;  // Accumulateur pour stocker les résultats des opé
   rations
unsigned int PC = PROGRAM_START;  // Compteur de programme, commence à l'
   adresse de début

// Fonction pour charger un programme en mémoire à partir d'un fichier
void charger_programme(const char *nom_fichier) {
    FILE *file = fopen(nom_fichier, "r");  // Ouvrir le fichier en mode
       lecture
    if (!file) {
        // Afficher un message d'erreur si le fichier ne peut pas être
           ouvert
        perror("Échec de l'ouverture du fichier");
        // EXIT_FAILURE vient de <stdlib.h>
        exit(EXIT_FAILURE);  // Quitter le programme en cas d'erreur
    }

    int adresse;
    unsigned char valeur1, valeur2;
    // Lire les valeurs du fichier et les stocker en mémoire
    // La notation &adresse, &valeur1, et &valeur2 utilise lopérateur
       dadresse (&) en C
    // Cet opérateur (&) est utilisé pour obtenir ladresse mémoire dune
       variable.
    while (fscanf(file, "%x %hhx %hhx", &adresse, &valeur1, &valeur2) == 3)
```

```
        {
        if (adresse >= MEMORY_SIZE - 1) {
             // Afficher un message d'erreur si l'adresse est invalide
            fprintf(stderr, "Adresse invalide : %x\n", adresse);
            fclose(file);  // Fermer le fichier
            exit(EXIT_FAILURE);  // Quitter le programme en cas d'erreur
        }
        // Stocker la première valeur à l'adresse spécifiée
        memory[adresse] = valeur1;
        // Stocker la deuxième valeur à l'adresse suivante
        memory[adresse + 1] = valeur2;
    }

    fclose(file);  // Fermer le fichier après la lecture
}

// Fonction pour exécuter le programme chargé en mémoire
void executer() {
    while (PC < MEMORY_SIZE) {
        unsigned char opcode = memory[PC++];  // Lire l'opcode à l'adresse
           actuelle du compteur de programme
        unsigned char operand;

        // Exécuter l'instruction en fonction de l'opcode
        switch (opcode) {
            case 0x00: printf("Instruction HALT rencontrée. Arrêt de l'exé
               cution.\n"); return;  // Arrêter l'exécution si l'opcode est
               0x00
            case 0x40: A += memory[PC++]; break;  // Ajouter la valeur
               suivante à l'accumulateur
            case 0x41: A -= memory[PC++]; break;  // Soustraire la valeur
               suivante de l'accumulateur
            case 0x48: A = memory[PC++]; break;   // Charger la valeur
               suivante dans l'accumulateur
            case 0x49: A &= memory[PC++]; break;  // ET logique avec la
               valeur suivante
            case 0x4A: A |= memory[PC++]; break;  // OU logique avec la
               valeur suivante
            case 0x4B: A ^= memory[PC++]; break;  // OU exclusif logique
               avec la valeur suivante
            case 0x4C: A = ~A; break;             // Négation binaire de l'
               accumulateur
            case 0x4D: A <<= 1; break;            // Décalage à gauche de l'
               accumulateur
            case 0x4E: A >>= 1; break;            // Décalage à droite de l'
               accumulateur
            case 0x4F: A += 1; break;             // Incrémenter l'
               accumulateur
            case 0x50: A -= 1; break;             // Décrémenter l'
               accumulateur
            case 0x51: A *= 2; break;             // Multiplier l'
               accumulateur par 2
            case 0x52: A /= 2; break;             // Diviser l'accumulateur
               par 2
            case 0x53: A &= 0xFF; break;          // ET logique avec 0xFF (
               aucun effet)
            case 0x54: A |= 0xFF; break;          // OU logique avec 0xFF (
               mettre tous les bits à 1)
            default: printf("Opcode inconnu : %02x\n", opcode); return;  //
               Afficher un message d'erreur pour un opcode inconnu
        }
```

```
        // Afficher l'état du compteur de programme et de l'accumulateur apr
           ès l'exécution
        printf("PC : %02x, A après exécution : %02x\n", PC - 1, A);
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        // Afficher un message d'utilisation si le nombre d'arguments est
           incorrect
        fprintf(stderr, "Utilisation : %s <fichier programme>\n", argv[0]);
        return EXIT_FAILURE;  // Quitter le programme en cas d'erreur
    }

    memset(memory, 0, MEMORY_SIZE);  // Initialiser la mémoire à zéro
    charger_programme(argv[1]);  // Charger le programme en mémoire

    // Débogage : Afficher le contenu de la mémoire avant l'exécution
    printf("Contenu de la mémoire avant exécution :\n");
    for (int i = 0; i < MEMORY_SIZE; i++) {
        printf("%02x ", memory[i]);
        if ((i + 1) % 16 == 0) printf("\n");
    }
    printf("\n");

    executer();  // Exécuter le programme
    printf("Valeur finale de A : %02x\n", A);  // Afficher la valeur finale
       de l'accumulateur
    // Quitter le programme avec succès
    return EXIT_SUCCESS;
}
```

Pour compiler et exécuter le programme, merci de faire :

```
gcc -W cx25.0.c -o main
./main program.hex
```

```
~/Bureau/IMPERATIVE/codes/cx25/cx25.0 (0.107s)
gcc -W cx25.0.c -o main
```

```
~/Bureau/IMPERATIVE/codes/cx25/cx25.0 (0.089s)
./main program.hex
Contenu de la mémoire avant exécution :
00 76 00 00 00 00 00 00 00 00 00 00 00 00 00 00
78 88 88 7c 00 00 00 00 00 00 00 00 00 00 00 00
00 01 7e 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
73 8a 00 00 00 00 00 00 71 8c 00 00 00 00 00 00
49 70 40 70 48 71 48 72 00 5e 48 8d 10 74 40 71
72 84 40 70 48 71 00 6c 48 8d 10 74 41 71 10 6e
00 71 00 00 74 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

PC : 51, A après exécution : 00
PC : 53, A après exécution : 70
PC : 55, A après exécution : 71
PC : 57, A après exécution : 72
Instruction HALT rencontrée. Arrêt de l'exécution.
Valeur finale de A : 72
```

5.2 cx25.1 : Coder la boucle d'évaluation pour en faire un STEPPER

Coder la boucle d'évaluation pour en faire un STEPPER qui affiche la valeur de PC et de A, et attend confirmation au clavier pour exécuter l'instruction suivante !; développez-le pour émuler celles des fonctionnalités de gdb qui vous paraissent utiles ou faciles.

Ce petit programme écrit en C simule un processeur simple avec une mémoire de taille fixe et un ensemble d'instructions de base. Il charge un programme à partir d'un fichier (program.hex), l'exécute et affiche l'état de l'accumulateur (A) et du compteur de programme (PC) à chaque étape. A la différence de cx25.0, le programme attend confirmation au clavier pour exécuter l'instruction suivante.

Voici le code source :

```c
/*
Amélioration de l'émulateur avec une boucle d'évaluation interactive, où chaque
instruction nécessite une confirmation de l'utilisateur avant de passer à
l'instruction suivante. Ce mode permet de déboguer et de suivre l'exécution du
programme étape par étape, similaire à un mode "stepper" dans les débogueurs.

compilation: gcc -W cx25.1.c -o main
             ./main program.hex
*/


// Inclusion de la bibliothèque standard pour les entrées/sorties (printf, scanf, etc.)
#include <stdio.h>
// Inclusion de la bibliothèque standard pour les fonctions utilitaires générales (malloc, free, exit, etc.)
#include <stdlib.h>
 // Inclusion de la bibliothèque pour les opérations sur les chaînes de caractères (memset, memcpy, etc.)
#include <string.h>

// Définir la taille de la mémoire et l'adresse de début du programme
#define MEMORY_SIZE 256
#define PROGRAM_START 0x50  // Adresse de début du programme

// Déclaration de la mémoire et des registres
unsigned char memory[MEMORY_SIZE];
unsigned char A = 0;  // Accumulateur
// Compteur de programme, commence à l'adresse de début
unsigned int PC = PROGRAM_START;

// Fonction pour charger un programme en mémoire à partir d'un fichier
void charger_programme(const char *nom_fichier) {
    FILE *file = fopen(nom_fichier, "r");
    if (!file) {
        perror("Échec de l'ouverture du fichier");
        exit(EXIT_FAILURE);
    }

    int adresse;
    unsigned char valeur1, valeur2;
    // Lire les valeurs du fichier et les stocker en mémoire
    while (fscanf(file, "%x %hhx %hhx", &adresse, &valeur1, &valeur2) == 3)
       {
        if (adresse >= MEMORY_SIZE - 1) {
            fprintf(stderr, "Adresse invalide : %x\n", adresse);
            fclose(file);
            exit(EXIT_FAILURE);
        }
        memory[adresse] = valeur1;
        memory[adresse + 1] = valeur2;
    }

    fclose(file);
}
```

```c
// Fonction pour exécuter le programme chargé en mémoire
void executer() {
    while (PC < MEMORY_SIZE) {
        // Lire l'opcode à l'adresse actuelle du compteur de programme
        unsigned char opcode = memory[PC++];
        unsigned char operand;

        // Afficher l'état du compteur de programme et de l'accumulateur
           avant l'exécution
        printf("PC : %02x, A avant exécution : %02x\n", PC - 1, A);
        printf("Appuyez sur Entrée pour continuer...\n");
        getchar();  // Attendre une entrée de l'utilisateur

        // Exécuter l'instruction en fonction de l'opcode
        switch (opcode) {
            case 0x00: printf("Instruction HALT rencontrée. Arrêt de l'exé
               cution.\n"); return;
            case 0x40: A += memory[PC++]; break;  // Ajouter la valeur
               suivante à l'accumulateur
            case 0x41: A -= memory[PC++]; break;  // Soustraire la valeur
               suivante de l'accumulateur
            case 0x48: A = memory[PC++]; break;   // Charger la valeur
               suivante dans l'accumulateur
            case 0x49: A &= memory[PC++]; break;  // ET logique avec la
               valeur suivante
            case 0x4A: A |= memory[PC++]; break;  // OU logique avec la
               valeur suivante
            case 0x4B: A ^= memory[PC++]; break;  // OU exclusif logique
               avec la valeur suivante
            case 0x4C: A = ~A; break;             // Négation binaire de l'
               accumulateur
            case 0x4D: A <<= 1; break;            // Décalage à gauche de l'
               accumulateur
            case 0x4E: A >>= 1; break;            // Décalage à droite de l'
               accumulateur
            case 0x4F: A += 1; break;             // Incrémenter l'
               accumulateur
            case 0x50: A -= 1; break;             // Décrémenter l'
               accumulateur
            case 0x51: A *= 2; break;             // Multiplier l'
               accumulateur par 2
            case 0x52: A /= 2; break;             // Diviser l'accumulateur
               par 2
            case 0x53: A &= 0xFF; break;          // ET logique avec 0xFF (
               aucun effet)
            case 0x54: A |= 0xFF; break;          // OU logique avec 0xFF (
               mettre tous les bits à 1)
            default: printf("Opcode inconnu : %02x\n", opcode); return;
        }

        // Afficher l'état du compteur de programme et de l'accumulateur apr
           ès l'exécution
        printf("PC : %02x, A après exécution : %02x\n", PC - 1, A);
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Utilisation : %s <fichier programme>\n", argv[0]);
        return EXIT_FAILURE;
    }
```

```
    // Initialiser la mémoire à zéro
    memset(memory, 0, MEMORY_SIZE);
    // Charger le programme en mémoire
    charger_programme(argv[1]);

    // Débogage : Afficher le contenu de la mémoire avant l'exécution
    printf("Contenu de la mémoire avant exécution :\n");
    for (int i = 0; i < MEMORY_SIZE; i++) {
        printf("%02x ", memory[i]);
        if ((i + 1) % 16 == 0) printf("\n");
    }
    printf("\n");

    // Exécuter le programme
    executer();
    // Afficher la valeur finale de l'accumulateur
    printf("Valeur finale de A : %02x\n", A);
    // Quitter le programme avec succès
    return EXIT_SUCCESS;
}
```

Il ne reste plus qu'à compiler et exécuter notre programme :

```
gcc -W cx25.1.c -o main
./main program.hex
```

```
~/Bureau/IMPERATIVE/codes/cx25/cx25.1 (0.134s)
gcc -W cx25.1.c -o main
```

```
~/Bureau/IMPERATIVE/codes/cx25/cx25.1 (24.808s)
./main program.hex
Contenu de la mémoire avant exécution :
00 76 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 88 7c 00 00 00 00 00 00 00 00 00 00 00 00
00 01 7e 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
71 7a 00 00 00 00 00 00 71 80 00 00 00 00 00 00
49 70 40 70 48 71 48 72 00 5e 48 8d 10 74 40 71
48 72 40 70 48 71 00 6c 48 8d 10 74 41 71 10 6e
00 71 00 00 74 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

PC : 50, A avant exécution : 00
Appuyez sur Entrée pour continuer...

PC : 51, A après exécution : 00
PC : 52, A avant exécution : 00
Appuyez sur Entrée pour continuer...

PC : 53, A après exécution : 70
PC : 54, A avant exécution : 70
Appuyez sur Entrée pour continuer...

PC : 55, A après exécution : 71
PC : 56, A avant exécution : 71
Appuyez sur Entrée pour continuer...

PC : 57, A après exécution : 72
PC : 58, A avant exécution : 72
Appuyez sur Entrée pour continuer...

Instruction HALT rencontrée. Arrêt de l'exécution.
Valeur finale de A : 72
```